GRAPE-8 — An Accelerator for Gravitational N-body Simulation with 20.5Gflops/W Performance

Junichiro Makino
Interactive Research Center of Science
Tokyo Institute of Technology
2-12-1 Ookayama, Meguro, Tokyo 152-8551 JAPAN
Email: makino@geo.titech.ac.jp

Hiroshi Daisaka
Graduate School of Commerce and Management,
Hitotsubashi University, Tokyo 186-8601, Japan
Email: daisaka@phys.science.hit-u.ac.jp

***Abstract*—In this paper, we describe the design and performance of GRAPE-8 accelerator processor for gravitational N-body simulations. It is designed to evaluate gravitational interaction with cutoff between particles. The cutoff function is useful for schemes like TreePM or Particle-Particle Particle-Tree, in which gravitational force is divided to short-range and long-range components. A single GRAPE-8 processor chip integrates 48 pipeline processors. The effective number of floating-point operations per interaction is around 40. Thus the peak performance of a single GRAPE-8 processor chip is 480 Gflops. A GRAPE-8 processor card houses two GRAPE-8 chips and one FPGA chip for PCI-Express interface. The total power consumption of the board is 46W. Thus, theoretical peak performance per watt is 20.5 Gflops/W. The effective performance of the total system, including the host computer, is around 5Gflops/W. This is more than a factor of two higher than the highest number in the current Green500 list.**

I. Introduction

A. Gravitational N-body problem

Many astronomical objects, in the first approximation, consist of many particles interacting with each other through gravity. In other words, they are gravitational N-body systems. Examples of such systems include our solar system, star clusters, galaxies, clusters of galaxies, and large scale structure of the universe. Very roughly speaking, anything bigger than a single star is a self-gravitating N-body system. In some cases, the presence of gas, such as the interstellar gas in the galaxy, is important. In some other cases, for example in the study of the planetary formation process, the finite size of "particles" plays important roles. In all cases, however, the gravity plays the primary role.

In the case of simulation of a galaxy, we need to model dark matter, stars, and interstellar gas. The most widely used approach is to express all of dark matter, stars, and gas by particles, using SPH (Smoothed Particle Hydrodynamics) method for hydrodynamic interaction between the gas particles. In this case, the calculation cost of gravity is generally higher than that of SPH interaction for two reasons. First, we need to calculate the SPH interaction only for gas particles, which is a small fraction of all particles in the system. Second, hydrodynamics interaction is of short-range nature, while gravity is a long-range force.

In the case of the simulation of planetary formation process, we need to model physical collisions between planetecimals. At present, almost all simulations are done with the simplified assumption that, when two planetecimals collide, they merge (perfect accretion). Of course, this assumption is far from accurate, and it is necessary to calculate the outcome of a collision between two planetecimals in some way. Such a simulation is however yet to be done. One reason is that if we include the disruption due to collision, the number of planetecimals would increase rapidly, and calculation cost would become too high.

Thus, in both cases, the calculation cost of gravitational interaction limits the number of particles, and thus resolution, of the simulation.

There are several possible approaches to speed up the calculation, and the following three approaches are probably the most important. The first one is to reduce the calculation cost by using sophisticated algorithms. The second one is to use faster computers, which means to parallelize the simulation program. The third approach is to develop a computer specialized to this problem. In the following, we briefly overview these three approaches.

B. Tree algorithm and FMM

For the calculation of the gravitational interaction, the calculation cost is $O(N^2)$ per timestep for a naive approach, in which the gravitational force on a particle is calculated as the summation of forces from all other particles in the system. There are several algorithms which reduces the calculation cost to $O(N \log N)$ or even $O(N)$. The most widely used algorithm is the Barnes-Hut tree algorithm [1]. In this scheme, the particles are organized into an oct-tree structure, which is constructed by recursively dividing a cube into eight subcubes. The force on a particle from particles in a box is replaced by the multipole expansion, if the particle and the box is "well separated" in view of the accuracy. If not, force is evaluated by taking summation of eight forces from eight subcubes. Here again, the force is defined recursively. The force from the entire system can be evaluated by the same recursive procedure. The calculation cost of Barnes-Hut tree algorithm is $O(N \log N)$. Using the fast multipole method (FMM) [2], it is possible to reduce the cost to $O(N)$. The

SC12, November 10-16, 2012, Salt Lake City, Utah, USA
978-1-4673-0806-9/12/$31.00 ©2012 IEEE

basic idea of FMM is not to evaluate the multipole expansion at the location of particles. Instead, we shift the multipole expansion to spherical harmonics expansion at the center of a box, and use this expansion to calculate the forces on all particles in the box. We can take the summation of the shifted spherical harmonics expansions of forces from multiple boxes and shift it down to subboxes, and thus reduce the calculation cost from $O(N \log N)$ to $O(N)$.

C. Individual Timestep Algorithm

Another important aspect of the calculation cost of the gravitational N-body problem is the variation of the timestep. In the case of a system like a galaxy, the orbital timescale of the stars near the center of the galaxy is much shorter than that of stars far away from the center. Thus, it is desirable to assign different timesteps to different particles. Moreover, since the gravitational force is a pure attracting force, there is no lower limit for the distance between two stars, and thus there is no lower bound for the size of the timestep. This situation is quite different from that of many other particle-based simulations. In the case of molecular dynamics, strong repulsive force limits the distance between two atoms. In SPH, the pressure term puts a similar limit.

The individual timestep algorithm [3] has been the standard method for the simulation of star clusters or planetary formation, for the last fifty years. The basic idea of the individual timestep algorithm is, as its name suggests, to assign each particle its own time t and timestep Δt. The particle with the smallest "next time", $t + \Delta t$, is integrated, using the predicted positions of all other particles at its time to calculate the force at its time. Traditionally, variable-timestep predictor-corrector method had been used as the integration scheme, so that we can make "dense" prediction. Since early 1990s the Hermite scheme has become the standard, and so far the simplest 4th-order scheme [4] has been most widely used. Recently 6th-order scheme is becoming more popular [5].

D. The P^3T scheme

There have been early attempt to combine the tree or FMM algorithm and the individual timestep algorithm, but it turned out to be difficult to implement the combination of the tree or FMM algorithm and the individual timestep algorithm on large-scale parallel computers. This is not surprising, since the tree algorithm reduces the complexity from $O(N^2)$ to $O(N \log N)$, and the individual timestep algorithm reduces the number of particles which share the same time to a small fraction of total number of particles. Thus, the degree of parallelism available to the hardware is rather small. Recently, we developed a novel way to combine the two algorithms [6]. In our scheme, which we named $\mathrm{P^3T}$ (particle-particle particle-tree), the gravitational interaction between two particles is divided into short-range and long-range terms, in the same way as in the case of $\mathrm{P^3M}$ (particle-particle particle-mesh) [7] or PME [8] (particle-mesh Ewald) scheme. In the case of $\mathrm{P^3M}$ or PME, the long-range force is evaluated with FFT and short-range force is calculated directly. With these schemes, generally the second-order leap-frog scheme is used and the stepsizes for short-range and long-range forces are both constant.

In our $\mathrm{P^3T}$ scheme, we evaluate the long-range force using tree, and apply the leapfrog integrator with the constant stepsize, while we integrate the short-range interaction using the individual timestep algorithm with direct summation. By taking the effective distance of the short-range force comparable to the typical inter-particle distance, we can make the calculation cost of the individual timestep part small. On the other hand, the procedure of the tree part is the same as that of the simple implementation of the tree algorithm with constant timestep, and therefore easy to implement on massively parallel computers. At present, we believe our $\mathrm{P^3T}$ scheme is the most practical way to combine the tree and individual timestep algorithms.

E. Parallelization and special-purpose hardware

The second approach is to use fast computers, and this means at present that the program must be efficiently parallelized on large-scale parallel machines with the distributed-memory architecture. In the case of the tree algorithm with a constant, shared timestep, there is a long history of parallelization, and highly scalable implementations exist [9]–[11]. Very recently, effective use of accelerator hardware such as GPGPUs and SIMD execution units of CPUs has become critical. Hamada et al. [12] demonstrated an impressive performance on a large GPU cluster with 576 NVIDIA GPU cards. The parallelization algorithm is a modification of the method described in [13].

The third approach, to develop specialized hardware for N-body simulations, is what we have been working on for more than 20 years [14], [15]. The basic idea of our approach is to design and build an attached processor which performs the calculation of interactions between particles. In the case of the direct summation scheme, a large fraction of the CPU time is spent for the calculation of the interactions between particles. The calculation cost of the force on a particle is $O(N)$, while the cost of the rest of the calculation is $O(1)$. For $N \sim 100$, force calculation typically accounts for more than 90% of the total CPU time, and for $N \sim 10^4$ more than 99.9%. Thus, one can achieve a very large speedup by just accelerating the interaction calculation.

In the case of the tree algorithm, specialized hardware is not very useful for a naive implementation, since the calculation cost of the tree traversal is not negligible. Barnes [16] invented an algorithm suitable for specialized hardware. In his algorithm, tree traversal is done for a group of particles, and the list of particles and nodes which exert the force to a group of particles is constructed. Then we can evaluate the forces from the list to the group, using specialized hardware. Barnes introduced this algorithm to improve the performance of tree algorithm on the CDC Cyber-205 vector processor. At present, this algorithm is used also on scalar processors, since the original algorithm is not efficient on modern pipelined processors with SIMD extensions.

In the case of FMM, the most expensive parts of the calculation are the shifting of the multipole expansion to the local spherical harmonics expansion and the evaluation of interaction with nearby particles. These two parts have roughly equal contribution, when the calculation is optimized. Thus, if we can do the particle-particle interaction very fast, we can reduce the total calculation time, asymptotically as square root of the speed up factor of the particle-particle interaction. In addition, it is also possible to express the multipole expansion by pseudoparticles and spherical harmonics expansions by the values of potential at discrete points [17], [18]. This and similar techniques are widely used in recent large-scale parallel implementations of FMM. Thus, FMM can be efficiently accelerated by specialized hardware for particle-particle interaction.

F. GRAPE systems

We have been developing special-purpose processors for the calculation of particle-particle interaction since 1989. We call these processors GRAPE (GRAvity PipE). The basic idea is to develop a fully pipelined processor for the calculation of the gravitational force between particles. There are several important advantages of this approach compared to usual software solutions. First, hardwired pipeline is extremely efficient in transistor usage. A modern scalar microprocessor core requires hundreds of millions of transistors, when we include the transistors for cache memories, and yet can perform less than ten floating-point operations per clock cycle. On the other hand, a double-precision adder/subtractor requires around 30k transistors, a single-precision multiplier also around 30k, a double-precision multiplier around 100k. Thus, if we construct a pipelined processor with relatively small number of double-precision multipliers, it requires two orders of magnitude smaller number of transistors per operation per clock. If the required accuracy is lower than that of the standard single-precision arithmetic, the advantage of a pipelined processor is even larger.

Second advantage is that memory access is significantly reduced. A single pipeline performs around 40 operations for four words (three components of the position vector and the mass of particles). While in operation, one pipeline calculates and accumulates the forces from many particles to one particle. Multiple pipelines can share one memory unit. since they can calculate the forces to multiple particles, from a single set of particles, as we have seen for the tree algorithm or FMM. It is also possible to let single pipeline calculate the forces on multiple particles. Thus, we can further reduce the requirement for the memory bandwidth. Of course, this small requirement helps us to achieve high efficiency on general-purpose microprocessors. However, the design of microprocessors is not optimized for very small bandwidth requirement. Thus, a significant fraction of the hardware is not used.

The third advantage is that overhead caused by using large number of arithmetic units is pretty small. When we try to use large number of microprocessors to calculate the force on one particle, there will be huge overhead due to communication latency. With special-purpose hardware, the latency is very small. When we use multiple pipelines to calculate forces on multiple particles from the same set of particles, the data of one memory unit must be broadcasted to multiple pipelines. This broadcast is done through simple fan-out tree (with pipeline stages if necessary). Thus it adds just a few clock cycles to the initial start up latency. We can also use multiple pipelines to calculate the force on one particle, and in this case it is necessary to take the summation after each pipeline finishes the calculation. This can be done with hardwired reduction tree, again adding several clock cycles. Thus, we can use thousands or more arithmetic units with negligibly small overhead.

First several systems were constructed using commercially available large-scale integrated circuits (LSIs). In GRAPE-1 and GRAPE-1A [19], [20], the logarithmic format was used instead of floating-point format, and arithmetic operations were implemented using EPROM chips, except for the first subtractions of position vector and final accumulations, for which integer ALU chips were used. In GRAPE-2, GRAPE-2A and HARP-1 [21]–[23], single-precision floating-point LSI chips were used for most of operations, while double-precision chips were used for initial subtraction and final accumulation.

With GRAPE-3 [24], we started to develop custom LSI chips in which we integrate pipelined processors. GRAPE-3 and 4 are, roughly speaking, single-chip implementation of GRAPE-1 and 2. One pipeline processor is integrated into one custom LSI, and multiple pipeline chips on one fairly large printed-circuit board (PCB) share one memory unit. In the case of GRAPE-3, 24 chips share one memory unit. In the case of GRAPE-4 [25], 47 chips share one memory unit. In addition, GRAPE-4 is designed so that up to nine processor boards are connected to a controller board which performs the reduction of the results from nine boards. The peak speed of GRAPE-4 is 1.08 Tflops, for the power consumption less than 25kW, at the year 1995.

GRAPE-5 and 6 are successor of GRAPE-3 and 4, integrating multiple pipelines into one chip [26], [27]. GRAPE-5 and 6 integrate 2 and 6 pipelines, respectively. In the case of GRAPE-6, 32 chips on one processor board are connected to broadcast/reduction networks, and four processor boards are connected to broadcast/reduction networks implemented on another board (network board). In addition, broadcasting can be done over multiple network boards. GRAPE-6 with 64 processor boards achieved the theoretical peak speed of 64 Tflops with the power consumption of around 40kW, at the year of 2002. The performance-per-watt number of GRAPE-6 was nearly two orders of magnitude better than that of microprocessor-based systems of that time.

After GRAPE-6, however, we were not able to develop a new machine with pipelined processors integrated to full-custom LSI chips. The reason is simply that the initial development cost of a full-custom chip has become too high. Roughly speaking, the initial cost of a custom chip has been proportional to the inverse of the feature size for the last two decades, for whatever reason. With 90nm technology, initial

cost exceeds two million USD, which was too high for a machine just for astrophysical N-body simulations. Thus, it has become necessary either to widen the application range or reduce the initial cost.

We moved to GRAPE-DR [28], which is a highly-parallel programmable SIMD processor. It achieved fairly high performance with low power consumption, around 4Gflops/W. This number is nearly an order of magnitude higher than that of microprocessors at the time, but not a great improvement over the GRAPE-6 processor chip.

Kawai and Fukushige developed GRAPE-7 [29], which is based on Cyclone-III FPGA (field-programmable gate array) chips from Altera. By adopting low-accuracy design similar to that of GRAPE-5, GRAPE-7 achieved pretty high performance-per-watt number, but with rather high production cost because large FPGA chips are pretty expensive.

G. Structure of this paper

In this paper, we describe the design, implementation, and the performance of GRAPE-8, the latest incarnation of GRAPE systems. With GRAPE-8, we used the structured ASIC (application-specific integrated circuit) technology from eASIC corp., which falls somewhere in between FPGA chips and custom LSI chips. The basic idea of the structured ASIC is to make an FPGA-like array of logic cells with programmable interconnection network, but using one layer of metal wiring to actually program the logic cells and interconnection network. In the case of an FPGA chip, both the logic cells and the interconnection networks are programmed with SRAM-based switches, resulting in low silicon usage, slow switching speed and high power consumption. In theory, a design based on structured ASIC technology can offer significantly higher silicon efficiency and thus higher performance-per-watt compared to FPGA chips, with initial cost much lower than that of custom LSI chips.

In section 2 we summarize the eASIC Nextreme-2 chip we used for GRAPE-8. In section 3 we describe the function and design of GRAPE-8 processor chip, and in section 4 the design of the processor board. GRAPE-8 processor pipeline is designed to calculate the gravitational interaction with or without cutoff, so that it can be used for schemes like $\mathrm{P^3T}$ or $\mathrm{P^3M}$. One chip integrates 48 pipeline processors, which share one memory unit for particles which exert forces. The memory unit is also integrated into the chip. The number format is 32-bit fixed point for position, 64-bit fixed point for the final accumulation, and floating-point format with 9-bit mantissa for all other operations. The clock frequency is 250 MHz and each pipeline executes 40 floating-point operations per clock cycle. Thus, peak performance of one chip is 480 Gflops. The processor board is a standard short-size PCI-Express card with Gen1 x8 interface. The PCI-Express interface and control logics for GRAPE-8 chips are implemented on an Altera Arria GX FPGA chip.

In section 5 we describe the measured performance. The asymptotic communication speed with the host computer is currently around 1.2GB/s. At present, half of the theoretical peak performance is achieved for direct summation with $N = 20,000$. The power consumption is 46W (26W for GRAPE-8 chips and around 20W for the FPGA chip). Thus, the processor chip itself achieved 37Gflops/W, while the board achieved 21Gflops/W. Section 6 is for discussions.

II. Nextreme-2 Structured ASIC device

The Nextreme-2 device of eASIC is its second-generation product, manufactured with 45-nm CMOS process. It integrates up to 737k logic elements and 11.5Mbits of DRAM. A logic element consists of two 3-input lookup tables. 576 logic elements and 384 DFFs form one block called eUnit, and four eUnits, one 36-Kbit memory (up to 36-bit width), two 512-bit register files (up to 36-bit width) forms one eGroup. One chip integrates up to 320 eGroups, and we used this largest chip, N2X740, as the device for GRAPE-8.

The logic elements of most of modern FPGAs contain full adders, and some of large FPGAs integrate hardwired multipliers. These specialized logic improve the performance of FPGAs for design like GRAPE processor considerably, since they are fast, compact, and low-power. The Nextreme-2 chip contains no such specialized logic. Another difference is that there is only a small number of small memory blocks in Nextreme-2. With FPGA chips, we can use the SRAM unit of logic elements as small SRAM memory. The logic elements of Nextreme-2 devices cannot be used as memory. Thus, though the number of logic elements is large, the performance of Nextreme-2 is not so high for arithmetic operations.

III. The design of GRAPE-8 chip

In this section, we describe the design of the GRAPE-8 processor chip. First, we describe what is calculated, and then the structure of the processor chip from the top level to lower.

A. Functionality

GRAPE-8 processor is designed to evaluate acceleration and potential at the position of particle i due to other particles.

$$\mathbf{a}_i = \sum_j G m_j g(\eta_i r_{ij}/r_0) \frac{\mathbf{r}_{ij}}{(r_{ij}^2 + \epsilon_i^2 + \epsilon_j^2)^{3/2}}, \quad (1)$$

$$\phi_i = \sum_j G m_j g(\eta_i r_{ij}/r_0) \frac{1}{(r_{ij}^2 + \epsilon_i^2 + \epsilon_j^2)^{1/2}}, \quad (2)$$

where

$$\mathbf{r}_{ij} = \mathbf{x}_j - \mathbf{x}_i, \quad (3)$$

and $\mathbf{x}_i$ is the position of particle i, G is the gravitational constant, m_j is the mass of particle j, ϵ is the softening parameter used to avoid the divergence at $r_{ij} = 0$, $g(r)$ is an arbitrary function, r_0 is the scale length. It is possible to use particle-dependent scale length by changing the value of η_i. Note that we use the same cutoff function g for both the force and potential. Thus, they are not consistent, but that does not cause any problem since the potential is used only for diagnostics purpose.

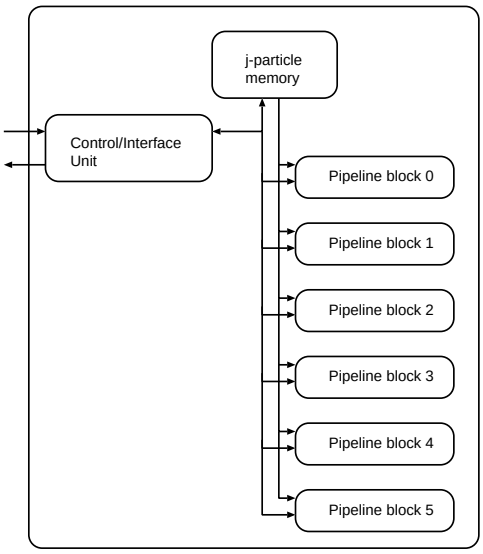

Fig. 1. The top-level structure of the GRAPE-8 processor chip

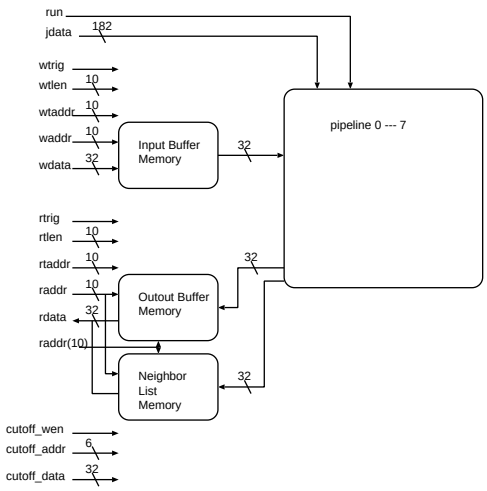

Fig. 2. The structure of the pipeline block

GRAPE-8 processor determines the index of the nearest neighbor, and also constructs a list of neighbor particles. The criterion for the neighbor is

$$r_{ij}^2 < h_i^2 + h_j^2. \tag{4}$$

B. The top-level structure

Figure 1 shows the top-level structure of the GRAPE-8 chip. It consists of a control/interface unit, a j-particle memory, and six pipeline blocks. The control unit handles the communication with the host computer through the interface FPGA chip. There are one input and one output port, each with 32-bit data width. They operate with the clock synchronized to the internal clock of 250MHz. Thus, the communication bandwidth is 1GB/s. The logic level is 1.8V SSTL (Stub series terminated logic), the logic level used for DDR2 memory interface.

In addition, the control unit generates control signals for the j-particle memory unit and pipeline blocks, according to the command input from the input port.

The j-particle memory stores the data of particles which exert gravitational forces. In this paper we call the particles that exert forces j-particles, and those receive the force i-particles. It is implemented using the block RAM units of the device. It is a dual-port memory with 16384-word depth and 182-bit width.

The pipeline blocks contain eight force-calculation pipelines, one neighbor-memory unit, and buffer memories for data transfer.

In the following, we describe the pipeline block, the j-particle memory unit, and the control/interface unit.

C. The pipeline block

Figure 2 shows the structure of the pipeline block. It consists of eight pipeline units, one input buffer memory, one output buffer memory, and one neighbor list unit. All memory units are dual-ported.

The purpose of the input/output buffer memory is to allow concurrent execution of calculation and data transfer. To start calculation, we first need to write the data of i-particles (particles which receive the gravitational force) to the input buffer memory, and then copy the data to the registers of the pipeline units. After the data are stored, the calculation is started. During the calculation, the buffer memory is not used, and we can write the data of the next set of i-particles to the buffer memory. The output buffer memory is used in a similar way, to allow data transfer concurrent with the calculation. In this approach, the data transfer is not completely hidden, since the transfer between the buffer memory and the pipeline unit cannot be overlapped with the calculation. It is possible to completely hide the communication by duplicating the registers within the pipeline unit. We decided not to do so since that approach would require a large number of DFF units.

D. The force calculation pipeline

Figure 3 shows the structure of the pipeline unit. This pipeline is designed to evaluate gravitational force and potential due to particle j at the position of particle i. The first step is to get the displacement vector using fixed-point subtractors and to convert the result to floating-point format. The position data $\mathbf{x}_i$ and $\mathbf{x}_j$ are expressed in 32-bit fixed-point format. The word length of 32 bits is long enough for practically all applications, because we intend to use GRAPE-8 with schemes like $\mathrm{P^3T}$ or TreePM, in which we apply cutoff function (either to the near side or far side) to the pure $1/r$ potential.

The floating-point format used in GRAPE-8 has 1-bit sign, 8-bit exponent and 9-bit mantissa. The value zero is expressed by a bit pattern of 0 for both the exponent and mantissa. The rounding mode is simple force-1 rounding, with modification to give unbiased result. When the bits lower than LSB before rounding are all zero, we do not set LSB to 1. In other cases,

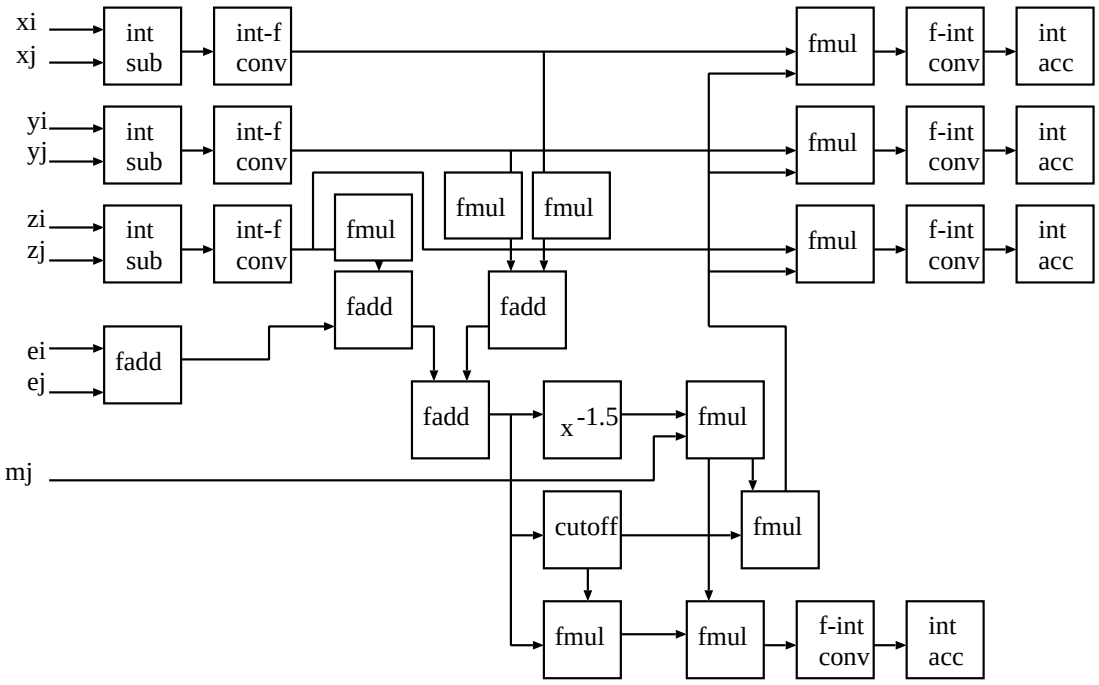

Fig. 3. The force calculation pipeline

we set LSB to 1. Compared to the usual round-to-nearest-even rounding, the advantage of unbiased force-1 rounding is just that it is significantly simpler to design and test. Of course, it requires one more bit to achieve the same level of accuracy, but additional amount of logic is small.

The cutoff function is implemented using table lookup and second-order polynomial interpolation. The table has 16 entries, and we supply $\eta_i r_{ij}^2/r_0^2$ converted to fixed-point format as the input to the cutoff function. We also need to calculate $r_{ij}^{-1.5}$, and this is done by table lookup and linear interpolation. The final accumulation is done in a 64-bit fixed-point format.

Each pipeline generates one-bit output flag which indicates particle j is within the specified distance $h_i^2 + h_j^2$ from particle i. The neighbor-list unit takes the logical OR of these flags from eight pipeline units and stores the index of particle j when the logical or of the flags is not zero.

E. j-particle memory unit

This unit is a 16384-word dual-port memory with 182-bit data width. The write port has 32-bit width. Usual random access is supported for write. For the read operation, burst access with arbitrary start address is supported. Thus, it is possible to send the new set of j-particles during the calculation of the force from the previous set of j-particles.

Note that the size of the j-particle memory unit does not limit the size of the problem we can solve. In most cases, we use the tree algorithm. Therefore, if the j-particle memory unit can keep the list of particles constructed on the host computer, that is sufficient. Even when the list is larger, we can divide the list to smaller sublists and calculate the forces from them, and add them up on the host. This would of course result in the increase of the communication cost, but that is small since here the calculation cost is already high.

F. Control/interface unit

The control/interface unit takes care of data I/O and generation of control signals to other units. It has one 32-bit input port and one 32-bit output port (each with 4-bit parity bits). For the input port, the address and data are multiplexed. The address for the output port is also supplied from the input port. Both input and output port support automatic address increment.

G. Physical design of the chip

GRAPE-8 processor chip is implemented on an eASIC N2X740 device. The usage of logic elements is around 50%, which looks a bit low. However, the usage of DFF units is 65%, which is fairly high. The worst-case clock frequency is 250MHz, and the estimated power consumption at the design time was 10W.

IV. The processor board

Figure 4 shows the processor board with two GRAPE-8 chips and one Altera Arria GX FPGA chip. All I/O pins of GRAPE-8 chips are connected to the interface FPGA chip. Figure 5 shows the photograph. The large chip at the center of the board is the interface FPGA, and two large chips at the both side of the interface FPGA are the GRAPE-8 processor chips. Small black chips are DC/DC converter modules.

We used the GPCIe IP of KFCR Corp for the PCI-Express interface. It offers Gen1 x8 interface on the first generation Arria GX chip.

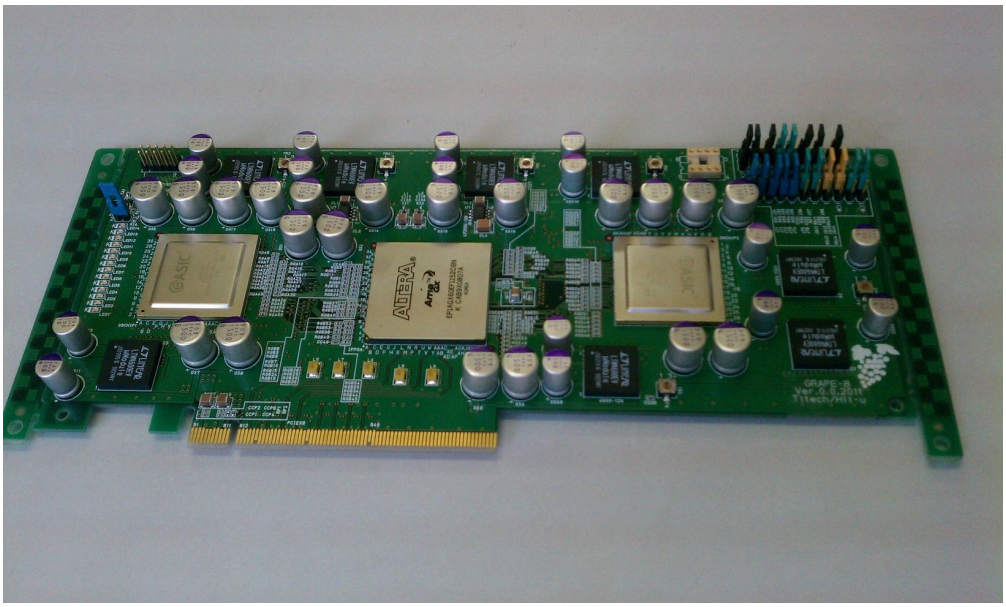

Fig. 5. The photograph of the processor board

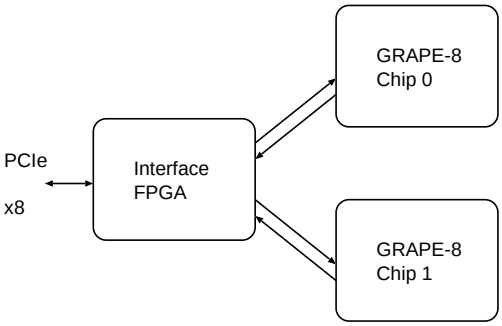

Fig. 4. The processor board

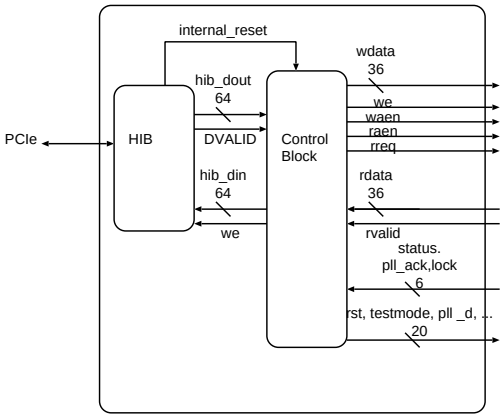

Fig. 6. The block diagram of the interface FPGA

Figure 6 shows the block diagram of the interface FPGA chip. It consists of the PCIe host interface block (HIB) and the control block. They are connected by 64-bit channels. The control block has external signals connected to GRAPE-8 chips. Thus, there is actually two sets of signals in the right-hand side of figure 6, for two GRAPE-8 chips.

V. Performance

In this section, we first describe the communication performance of the PCIe interface of the GRAPE-8 processor board. Then we describe the performance of the force calculation function.

A. Communication

Figure 7 shows the performance of data transfer between the GRAPE-8 processor board and the host computer. The CPU of the host computer is Intel Core i7-920 and the chipset is X58. The operating system is Scientific Linux 6.1.

The effective transfer rate of DMA write operation is very well reproduced by a simple model:

$$T_{DMA}(n) = T_{start} + nT_{byte}, \quad (5)$$

where $T_{DMA}(n)$ is the time to transfer n bytes, T_{start} is the start-up time of one DMA operation, and T_{byte} is the time to transfer 1 byte of data. The dashed curve in figure 7 is plotted with $T_{start} = 1.35\mu$s and $T_{byte} = 0.815$ns. Thus, the asymptotic transfer rate for large data block is 1.23 GB/s. This value is 61% of the theoretical maximum throughput of PCI-Express x8 interface(2GB/s). This is due to the packet-level overhead of PCI Express interface. The behavior of the DMA read operation is a bit more complex, since here the granularity of the maximum size of DMA read request (2KB in the present setting) is visible. As we will see in the next section, the performance of DMA operation is sufficient.

B. Force calculation

Figure 8 shows the calculation speed as the function of the number of particle N, for the calculation of forces on N particles from N particles. For this test, we used two chips of one board to calculate the forces on the same set of

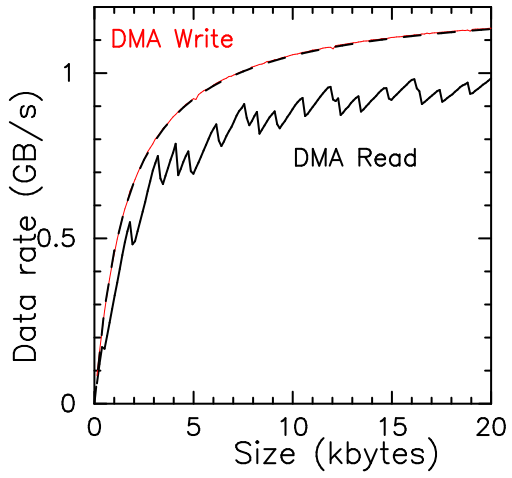

Fig. 7. The data transfer rate of DMA read and write operations in GB/s plotted as function of the data size in KB. Solid black and thin red curves show the speed of read and write operations, respectively. Thick dashed curve shows model fit for the write performance.

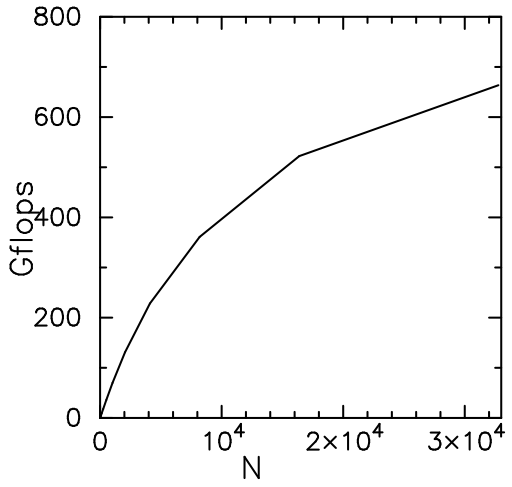

Fig. 8. The effective calculation speed for force calculation in Gflops as the function of number of particles N.

particles from different set of particles (j-parallel calculation). At present, we need fairly large value of N to obtain a reasonable fraction of the peak speed. Half of the peak speed is achieved for $N \sim 20,000$.

The reason why we need rather large number of particles to achieve high performance is that at present the driver software and the control logic with FPGA is not highly optimized. Figure 9 shows the breakdown of the time needed to calculate the force on one particle. Sending one particle takes 0.15 μs, and the start-up overhead and receiving the calculated result adds up to around 0.3 μs. The size of the data for one i- or j-particle is 24 bytes, and that for the calculated result is 40 bytes. The asymptotic data transfer rate for DMA write is 1.2GB/s and that for read is 1.1GB/s. Thus, after sufficient optimization, sending one particle should take no more than 22 ns, and receiving result 33ns. Therefore, we should be able to improve the performance of the board for small values of N by

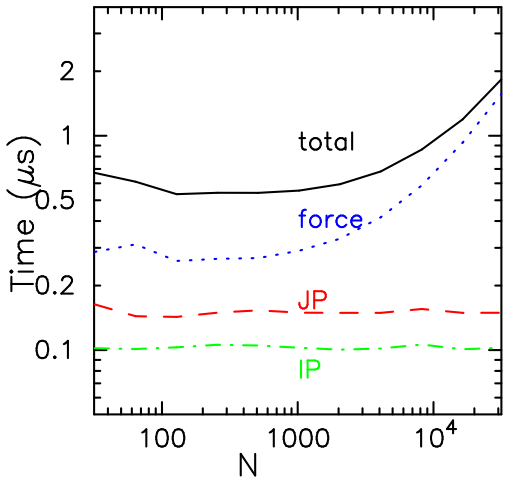

Fig. 9. The total time for one particle (solid black), the times spent for force calculation for one particle (dotted blue), sending one j-particle (red dashed), and sending one i-particle(green dot-dashed), plotted against the number of particles N.

at least a factor of 10. In addition, the GRAPE-8 processor chip is designed so that communication and calculation can proceed concurrently. We expect to achieve near-peak performance for $N \sim 1000$. This number is small enough to achieve good performance when used with the tree algorithm.

VI. Discussion and Summary

A. Comparison with other machines/approaches

In this paper we described the design and performance of GRAPE-8, a special-purpose computer based on eASIC's structured ASIC technology. In this section, we briefly discuss how our approach is compared to other approaches. We consider (a) modern FPGAs, (b) GPGPUs, and (c) Anton.

As in the case of GRAPE-7 [29], one could use FPGAs to implement pipeline processors for low-accuracy force calculation. Here, the primary difference is in the price-performance ratio. According to the Altera Website, the price of Cyclone V device with 150K logic elements is around USD 250. The mass-production price of the eASIC N2X740 chip is about the same. Thus, the price per gate of the newest FPGA, with 28nm technology, is still several times higher than that of 45nm eASIC chip. We have not tried speed comparison, but we expect the operating frequency would be similar. Thus, if one plan to use more than a few hundred chips, structured ASIC seems to be a viable solution. The power consumption is similar for two devices.

The use of GPGPUs has been quite successful for N-body simulations, at least partly because of the algorithms developed for the efficient use of GRAPE hardware [12], [31]. The transistor efficiency is much higher for GRAPE-8, but because of the mass-production effect performance-per-dollar would be somewhat higher for the latest GPUs (such as HD6000 series from AMD). The performance per watt is much higher for GRAPE-8, even when compared with the latest GPUs.

Anton [30] is a special-purpose computer for fast molecular dynamics simulation with relatively small number of atoms (typically less than 10^5). The main target of Anton is millisecond-level simulation of protein molecules, and it can handle one timestep of MD simulation in about 10μs.

In order to achieve this performance, Anton combined GRAPE-like force-calculation pipelines with programmable processor cores, and connected many of them with a specialized low-latency torus network. Thus, Anton is a complex system which required at least one order of magnitude more cost and human resources to develop, compared to those for GRAPE-8. Anton's approach was necessary for the problem they want to solve. On the other hand, the main target of GRAPE-8 is large-N calculations with more than 10^6 particles. Thus, the need for extremely low latency is not so severe. We believe that the difference between the designs of GRAPE-8 and Anton comes from the difference in the target problems, and that it by no means indicates the superiority of one design to another.

B. Summary

In this paper, we describe the design, implementation and performance of the special-purpose computer for gravitational N-body problem, GRAPE-8. It is specialized for the calculation of particle-particle interaction, and fabricated using the Structured ASIC device Nextreme-2 N2X740 from eASIC. The processor chip integrates 48 pipeline processors which operate at the clock speed of 250MHz, and offers the peak speed of 480 Gflops. A processor board with two GRAPE-8 chips and one interface FPGA chip gives the speed of 960 Gflops for the power consumption of 46W. Thus it achieved the extremely high performance-per-watt number of 20.9 Gflops/W. The board works as designed, and we are currently working on the optimization of the data transfer software and control logic of the FPGA chip.

ACKNOWLEDGMENT

The authors would like to thank Satoshi Yasuda and others from Innotech Corp., who did the gate-level design and physical synthesis of the GRAPE-8 chip, and Toshiyuki Fukushige and Atsushi Kawai for providing the technical support for GPCIe. This work is supported by KAKENHI(21244020).

REFERENCES

[1] J. Barnes and P. Hut, "A hiearchical o(nlogn) force calculation algorithm," *Nature*, vol. 324, pp. 446–449, 1986.

[2] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 73, pp. 325–348, December 1987.

[3] S. J. Aarseth, *Gravitational N-Body Simulations*. Gravitational N-Body Simulations, by Sverre J. Aarseth, pp. 430. ISBN 0521432723. Cambridge, UK: Cambridge University Press, November 2003., Nov. 2003.

[4] J. Makino and S. J. Aarseth, "On a hermite integrator with ahmad-cohen scheme for gravitational many-body problems," *Publ. of the Astron. Society of Japan*, vol. 44, pp. 141–151, 1992.

[5] K. Nitadori and J. Makino, "Sixth- and eighth-order Hermite integrator for N-body simulations," *New Astronomy*, vol. 13, pp. 498–507, Oct. 2008.

[6] S. Oshino, Y. Funato, and J. Makino, "Particle-Particle Particle-Tree: A Direct-Tree Hybrid Scheme for Collisional N-Body Simulations," *Publ. of the Astron. Society of Japan*, vol. 63, pp. 881–893, Aug. 2011.

[7] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles*. Bristol: IOP Publishing, Ltd., 1988.

[8] T. Darden, D. York, and L. Pedersen, "Particle mesh Ewald: An $N \cdot \log(N)$ method for Ewald sums in large ssytems," *J. Chem. Phys*, vol. 98, pp. 10 089–10 092, 1993.

[9] J. K. Salmon, G. S. Winckelmans, and M. S. Warren, "Fast parallel treecodes for gravitational and fluid dynamical N-body problems," *Intl. J. Supercomputer Appl.*, vol. 8, pp. 129–142, 1994.

[10] J. Dubinski, "A parallel tree code," *New Astronomy, vol. 1, no. 2, p. 133-147.*, vol. 1, pp. 133–147, Oct. 1996, provided by the NASA Astrophysics Data System.

[11] V. Springel, N. Yoshida, and S. D. White, "Gadget: A code for collisionless and gasdynamical cosmological simulations," *New Astronomy*, vol. 6, pp. 79–117, 2001.

[12] T. Hamada and K. Nitadori, "190 tflops astrophysical n-body simulation on a cluster of gpus," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. Washington DC: IEEE, 2010, p. Online.

[13] J. Makino, "A Fast Parallel Treecode with GRAPE," *Publ. of the Astron. Society of Japan*, vol. 56, pp. 521–531, Jun. 2004.

[14] D. Sugimoto, Y. Chikada, J. Makino, T. Ito, T. Ebisuzaki, and M. Umemura, "A special-purpose computer for gravitational many-body problems," *Nature*, vol. 345, pp. 33–35, 1990.

[15] J. Makino and M. Taiji, *Scientific Simulations with Special-Purpose Computers — The GRAPE Systems*. Chichester: John Wiley and Sons, 1998.

[16] J. E. Barnes, "A modified tree code: don't laugh; it runs." *J. Comp. Phys.*, vol. 87, pp. 161–170, 1990.

[17] J. Makino, "Yet another fast multipole method without multipoles — pseudo-particle multipole method," *Journal of Computational Physics*, vol. 151, pp. 910–920, 1999.

[18] A. Kawai, J. Makino, and T. Ebisuzaki, "Performance Analysis of High-Accuracy Tree Code Based on the Pseudoparticle Multipole Method," *Astrophys. J.Supp.*, vol. 151, pp. 13–33, Mar. 2004.

[19] T. Ito, J. Makino, T. Ebisuzaki, and D. Sugimoto, "A special-purpose n-body machine grape-1," *Computer Physics Communications*, vol. 60, pp. 187–194, 1990.

[20] T. Fukushige, T. Ito, J. Makino, T. Ebisuzaki, D. Sugimoto, and M. Umemura, "Grape-1a: Special-purpose computer for n-body simulation with a tree code," *Publ. of the Astron. Society of Japan*, vol. 43, pp. 841–858, 1991.

[21] T. Ito, T. Ebisuzaki, J. Makino, and D. Sugimoto, "A special-purpose computer for gravitational many-body systems: Grape-2," *Publ. of the Astron. Society of Japan*, vol. 43, pp. 547–555, 1991.

[22] T. Ito, T. Fukushige, J. Makino, T. Ebisuzaki, S. K. Okumura, D. Sugimot, H. Miyagawa, and K. Kitamura, "A special-purpose computer for molecular dynamics: Grape-2a," *PROTEINS: Structure, Function, and Genetics*, vol. 20, pp. 139–148, 1994.

[23] J. Makino, E. Kokubo, and M. Taiji, "Harp: A special-purpose computer forn-body problem," *Publ. of the Astron. Society of Japan*, vol. 45, pp. 349–360, 1993.

[24] S. K. Okumura, J. Makino, T. Ebisuzaki, T. Fukushige, T. Ito, D. Sugimoto, E. Hashimoto, K. Tomida, and N. Miyakawa, "Highly parallelized special-purpose computer, grape-3," *Publ. of the Astron. Society of Japan*, vol. 45, pp. 329–338, 1993.

[25] J. Makino, M. Taiji, T. Ebisuzaki, and D. Sugimoto, "Grape-4: A massively parallel special-purpose computer for collisional n-body simulations," *Astrophys. J.*, vol. 480, pp. 432–446, 1997.

[26] A. Kawai, T. Fukushige, J. Makino, and M. Taiji, "Grape-5: A special-purpose computer for n-body simulations," *Publ. of the Astron. Society of Japan*, vol. 52, pp. 659–676, Aug. 2000.

[27] J. Makino, T. Fukushige, M. Koga, and K. Namura, "GRAPE-6: Massively-Parallel Special-Purpose Computer for Astrophysical Particle Simulations," *Publ. of the Astron. Society of Japan*, vol. 55, pp. 1163–1187, Dec. 2003.

[28] J. Makino, K. Hiraki, and M. Inaba, "Grape-dr: 2-pflops massively-parallel computer with 512-core, 512-gflops processor chips for scientific computing," in *Proceedings of SC07*. ACM, 2007, p. (Online).

[29] A. Kawai and T. Fukushige, "$158/gflop astrophysical n-body simulation with a reconfigurable add-in card and a hierarchical tree algorithm," ACM, p. (Online), 2006.

[30] D. E. Shaw, M. M. Deneroff, J. S. K. Ron O. Dror, R. H. Larson, C. Y. John K. Salmon, B. Batson, K. J. Bowers, M. P. E. Jack C. Chao, J. Gagliardo, C. R. H. J.P. Grossman, D. J. Ierardi,

J. L. K. István Kolossváry, T. Layman, M. A. M. Christine McLeavey, R. Mueller, Y. S. Edward C. Priest, J. Spengler, B. T. Michael Theobald, , and S. C. Wang, "Anton: A special-purpose machine for molecular dynamics simulation," in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA '07)*. ACM, 2007, pp. 1–12.

[31] J. Bédorf, E. Gaburov, and S. Portegies Zwart, "A sparse octree gravitational N-body code that runs entirely on the GPU processor," *Journal of Computational Physics*, vol. 231, pp. 2825–2839, Apr. 2012.